



Gannet: A Functional Programming Task Description for Service-based SoC Architectures

Wim Vanderbauwhede



Overview



- What is the Gannet Service-Based Architecture?
- Task description and task decomposition in Gannet
- Gannet and Functional Programming
- Conclusion



System-on-Chip with on-Chip Network

- With current IC technology, SoC's can be very big
- Traditional bus-style interconnect causes a bottleneck:
 - Synchronisation over large distances is impossible
 - Fixed point-to-point result in huge wire overhead
- On-chip networks provide a solution
 - globally asynchronous/locally synchronous
 - flexible connectivity



Service-Based Architecture - SoC/NoC

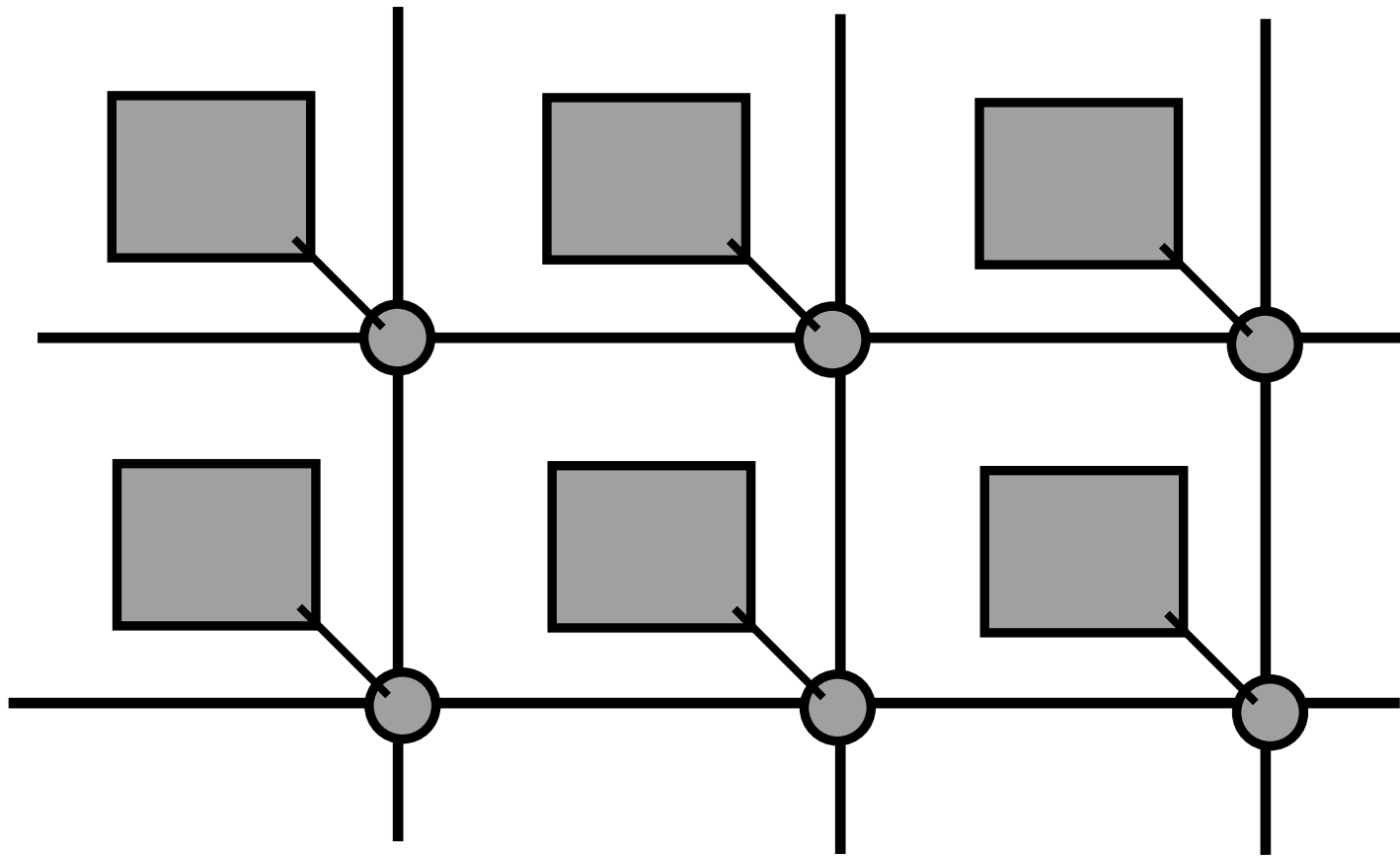


Typical Network on Chip

- Regular topology (tiles)
- Simple switches (self-routing)



Service-Based Architecture - SoC/NoC





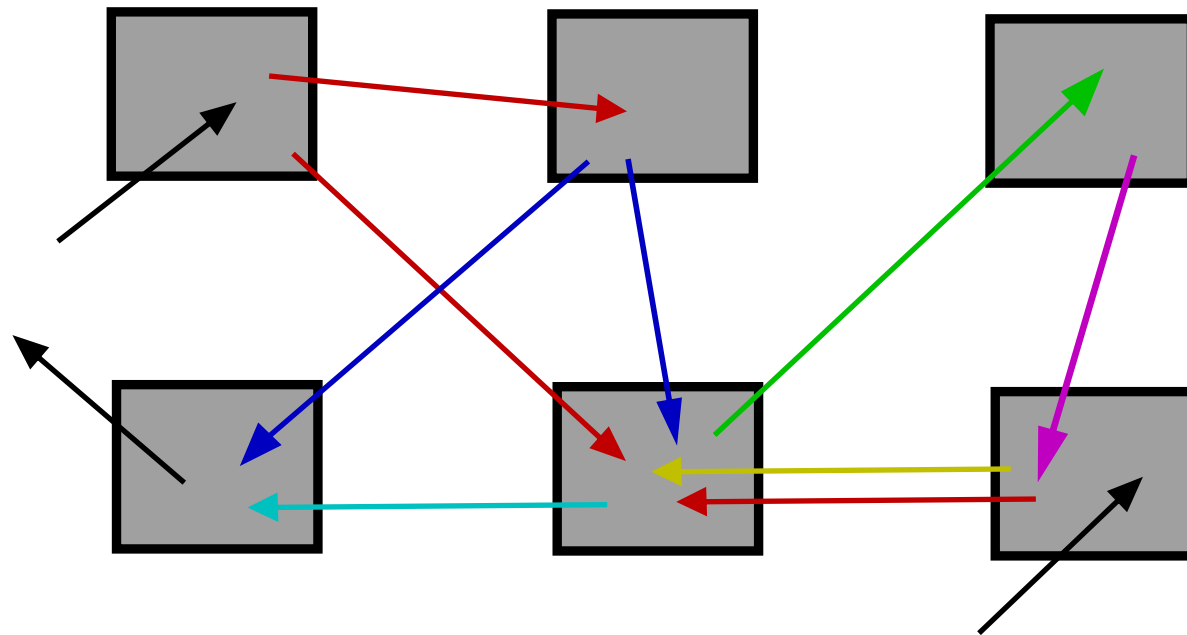
Service-Based Architecture - Concept



- The SoC architecture consists of a set of modules connected by the on-chip network
- Each module offers a particular service
- Tasks are performed by interacting sets of services



Service-Based Architecture - Concept

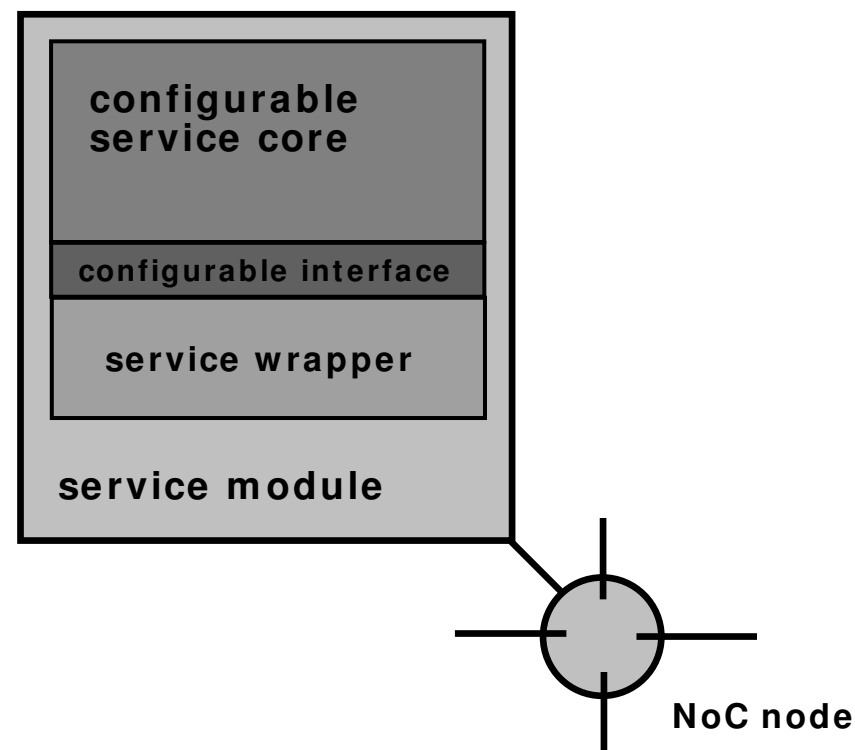




Service-Based Architecture - Service module



- A "service module" is a complex processing unit consisting of
 - a logic core: performs the actual data processing.
 - a "service manager": manages the flow of data to and from the core.





Service-Based Architecture - Properties



- Flexibility
 - many different tasks can be performed by a given set of services
 - services are themselves configurable
- Distributed processing
 - parallelism
 - globally asynchronous, event-driven
- Low overhead
 - service wrapper design is small compared to IP cores



SBA Task Management Strategies



- Services can be task-aware or task-unaware at instantiation
- Task-aware services require reconfiguration between different tasks, but no task-related information during the task
- Task-unaware services require no reconfiguration between different tasks, but need task-related information: a "task description"
 - A global task description formalism is used to define the task
 - Every service performs a number of subtasks
 - Services are memoryless with respect to the subtasks
- Task-unaware services can perform multiple tasks in parallel

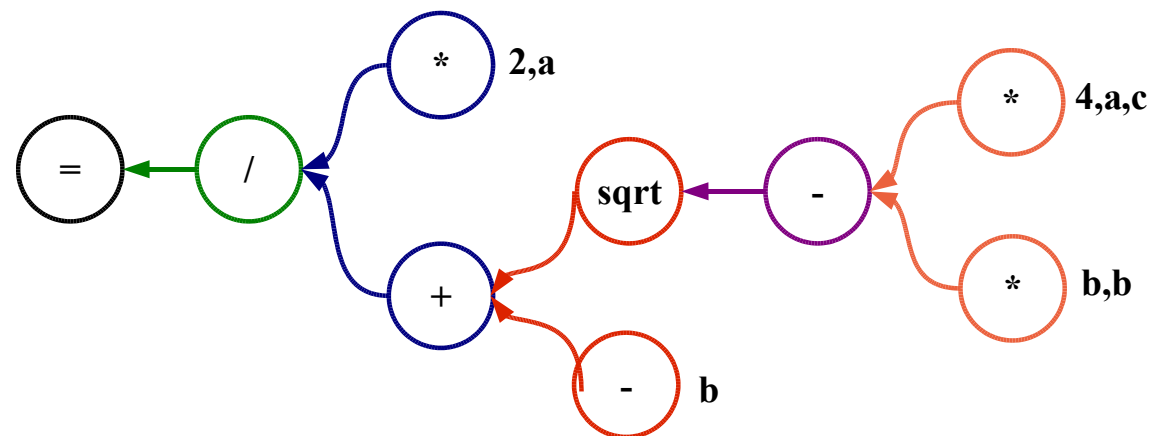


SBA Task Description (1)

Example task: calculate

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

for arbitrary values of a,b,c





SBA Task Description (2)



Task description format: S-expressions

Example:

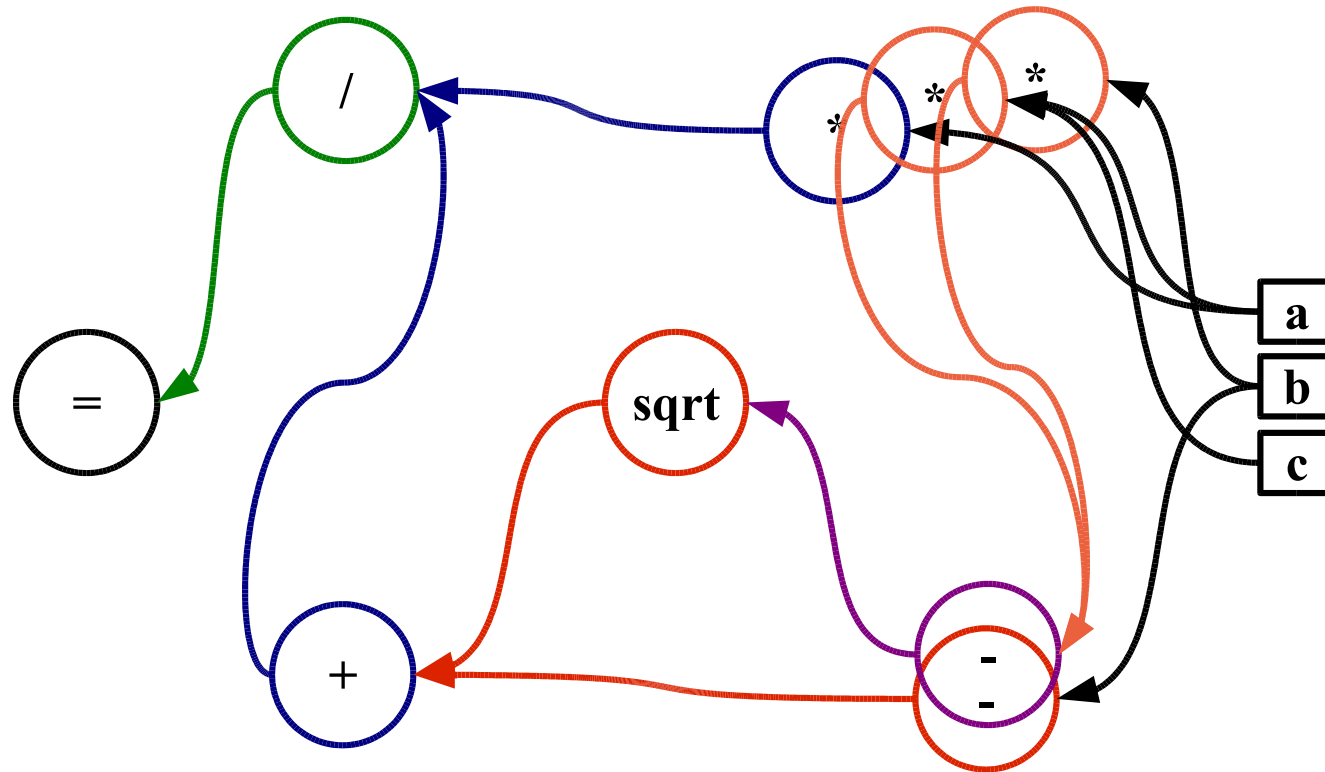
```
(/  
  (+  
    (- b)  
    (sqrt  
      (-  
        (* b b)  
        (* 4 a c)  
      ))  
    (* 2 a)  
  )  
)
```



SBA Task Description (3)



SBA view:





Gannet Task Decomposition



- Task descriptions enter the system via a gateway service
- The gateway
 - stores all data packets (variables)
 - passes task description on to first service
- The service wrappers
 - request required variables from the gateway
 - pass subtasks on to other services and wait for results
 - return results to the subtask sender



Task Description Limitations



- In general, the task description for a given set of services will not be Turing-complete:
 - Not powerful enough to implement algorithms
 - Guaranteed to halt for any particular description and input
- By adding language services, the system can be made Turing-complete
 - Main requirement: potential to create infinite loops
 - Consequence: need for recursion and branching



SBA and Functional Programming



- Services can be considered as functions
 - Same inputs result in same outputs
 - No side effects
- Task description is a pure functional language
 - pure: data packets ("variables") are read-only
 - functional: all tasks are expressed only in terms of functions (services) and immutable variables (data packets)
- In the SBA, all additions are services and, consequently, functions



Efficiency: Aliasing and combining services



- Language, arithmetical and logic services are small
 - Implementing them as individual physical services has a high overhead
 - But it allows fully parallel distributed processing
- Aliasing: Combining several abstract service in a single physical service
 - Reduce area overhead at cost of reduced parallelism
 - Extreme case: use a single ALU for all numerical and logic services



Conclusion



Summary

- A proposal to make the SBA Turing-complete using abstract services
- The resulting SBA language is a pure functional language
- SBA can be considered as a distributed computing platform which supports hardware implementations of functions